THE HORIZONTAL SPACING OF GRAPHICAL NOTATION

Mike Solomon

The University of Florida
School of Music, Gainesville, FL, USA
mike@apollinemike.com

ABSTRACT

While several articles have explored the automated spacing of a line of music, little attention has been paid to the automated spacing of graphical notation in the horizontal domain. This article proposes an algorithm to stretch and compress images so that they are able to be integrated into a traditional horizontal spacing model while still attaining certain "target points" in a score. This is done via a linear program that distributes stretching error over the graphical object. The linear program is described through a presentation of its constituent matrices along with several musical examples that demonstrate its functionality. The article concludes with ideas regarding the program's broader applicability to the domain of graphical notation.

1. INTRODUCTION

Since the ascendence of graphical notation in the 1960s, a major concern for composers and engravers alike has been the most efficient and musical manner to mix graphics with traditional notation. Thus far, three major approaches have emerged. The first, used by Mark Applebaum in his *On the Metaphysics of Notation*, is to create scores that make use of both traditional and unconventional glyphs without specifying ordinal relationships between these objects. A second approach, used throughout Cornelius Cardew's *Treatise*, is to use locally-traditional snippets of notation that are merged into a document with an unconventional layout. A third approach, used by Krzysztof Penderecki in his opera *Diably Z Loudun*, is the integration of non-traditional glyphs and images into the layout of a traditional score (time on the X-axis, instrument and pitch on the Y-axis).[1] Because of the highly individualized nature of these artworks, it is difficult to propose any unifying digital tool that can help composers automate certain aspects of their layout decisions. Headway has been made by libraries such as *Belle, Bonne, Sage* [3], which provides an easily-callable collection of rescalable musical glyphs. However, context-agnostic tools such as this do not draw upon the many musical engraving conventions to which performers have grown accustomed, demanding a significant coding overhead from the user wishing to mix received knowledge with experimental notation. This paper offers a partial solution to this problem, using a smoothing algorithm that adapts vector graphics to the dicta of conventional horizontal spacing as developed in [16], [18], [4], [8], [2], [9], [7], [17], and [1]. Said algorithm uses linear programming to constrain a graphic so that it attains certain horizontal "target points" along a line of music, smoothly distributing displacement error over the graphic so that it is minimally distorted from its original version and so that any distortion is not localized at the target points to which the graphic is anchored.

The paper will proceed as follows. First, it will survey canonical literature on horizontal spacing in engraving, according special attention to the algorithmic automation of this task. It will then propose a model for the horizontal spacing of graphical notation that uses data from this general spacing model. Finally, it will present examples of this algorithm's results and provide suggestions for future research.

2. HORIZONTAL SPACING

This section explores horizontal spacing from a traditional engraving approach. After surveying common horizontal spacing practices in music engraving, it then summarizes a popular horizontal spacing algorithm [8] that has been implemented by several digital engravers such as Lilypond [15] and GUIDO [10].

2.1. Traditional Horizontal Engraving Considerations

Horizontal spacing is traditionally calculated based on two principal constraints: fixed distances between glyphs (hereafter called 'rods') and variable distances between adjacent glyphs (hereafter called 'springs'). Ross [18] discusses the use of several of these rods in traditional music engraving, such as the standard distance between clefs and noteheads and the standard distance between clefs and time signatures (see Figure 1). Read [16] establishes a precursor to the present-day notion of springs through his charts of least-common-multiple relationships between durations (see Figure 2). Few canonical engraving texts, however, confront the issue of the area of music over which these calculations are relevant. Instead, they give examples that function within fixed widths that are often a measure long. Algorithmic horizontal engraving has sought to

[1] In this taxonomy, I am excluding works such as Stockhausen's *Zyklus*, which uses a system of organized glyphs that is intended to communicate precise musical information. In the present study, "graphical notation" refers to ambiguous notational schemes that are intended to evoke an interpretive response above and beyond that which "traditional" notation could presumably engender.

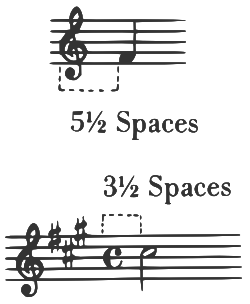

Figure 1. "Rod" spacing suggestions by Ross [18, p. 145].

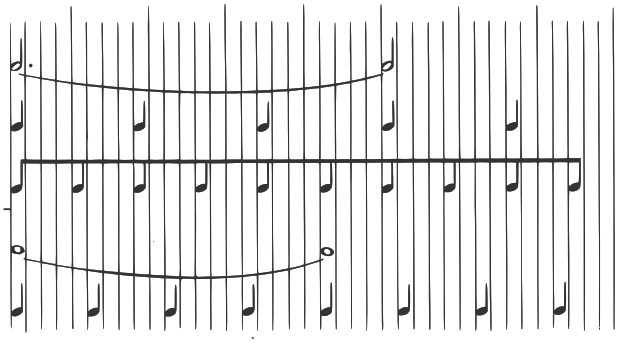

Figure 2. Reproduction of a horizontal spacing configuration showing a 5:8 metric division [16, p. 201]. This method of partitioning the horizontal space has been generalized via the [8] algorithm.

remedy this problem with a Spring-Rod model that can be applied to various lengths of staff space.

2.2. Algorithmic Horizontal Engraving

The issue of horizontal spacing in computer typesetting was first given comprehensive and publicly-available scholarly attention by Gourlay [8], who proposed a Box-Glue model[2] in which horizontal spacing of objects on lines of music is initiated by applying force to a line of notes with springs and rods between them and evaluating where these objects come to rest when the springs reach an equilibrium point after the force has dissipated. Haken and Blostein [9] introduce the terminology 'Spring' and 'Rod' to describe the Gourlay model, refining it to perform better in its handling of rods, and Renz [17] offers an improved algorithm that treats objects on the line as "neighborhoods" with flexible borders that can be adjusted over multiple passes of the Gourlay algorithm.

Currently, one of the more comprehensive versions of the algorithm, which takes into account all of the major research on the subject, is found in GNU Lilypond. Nienhuys [14] summarizes the algorithm as such:

- Each time-point gets a paper column that includes simultaneous events. For example, notes in a chord, articulations, dots, and accidentals all may fall into one column. This column takes into account polyphonic voices and multiple systems.
- The spacing engine computes an ideal for each column. This position takes the form of a spring of ideal length with a stretch factor to account for difference in fixed space (symbols) and stretchable space (white space between symbols) upon stretching or compressing.
- Springs are only implemented between adjacent columns. Each column pair has just one spring.
- Arbitrary (including non-adjacent) columns may specify minimum distances (rods) that represent the minimum widths necessary for preventing collisions.
- In monophonic situations, spacing is counted from the left edge of the note head to the next left-edge of the head (assuming that there are no chords with seconds or unmergeable unisons in them – otherwise, different spacing constants are applied that represent the expanded horizontal extent of the column).
- The common shortest note S in a piece is spaced with $2W$, where W is the width of a black notehead.
- Spaces are logarithmically assigned, so if S is the common shortest duration in the piece, then $D(2S) = 3W$, $D(4S) = 4W$ etc., where D is a function calculating the duration space.
- Calculate the shortest duration for each measure. The common shortest duration (*CSD*) is the one which appears in most measures. This is to prevent a single short note from stretching the entire piece.
- For durations R shorter than *CSD*, the space is $W + \frac{W \times R}{S}$.
- For polyphonic situations, the space is $\frac{DT}{SP} \times D(SP)$, where SP is the shortest note playing at that time point, and DT is the time difference between the columns.
- Spacing from the first clef to the first note retains a constant space (or rod) irrespective of the results above.
- The final result is achieved by applying force to final column in a line (for example, a double bar-line) and letting the force move through the springs until the upper and lower bound on the measures for a given system become equal. Optimal line breaks are calculated based on *CSD* as defined above.

Because this algorithm calculates line breaks based on the shortest duration in an entire work, it achieves a sense of global visual constancy that would not be possible if line break decisions were exogenous to the Spring-Rod model.

[2] This model is itself is based on the one developed for text layout in TEX [11].

The choice to do this is, of course, a subtle aesthetic one, and the unity it confers onto a piece's layout is not immediately perceptible to the musician reading through a work. However, a crucial argument in favor of this approach centers around spanners, a group of musical objects that is only cursorily treated in the above-cited articles on horizontal spacing. Spanners, such as *crescendo* and *diminuendo* markings, beams, slurs, and *ottava* indications, have the ability to reach over systems — and in certain extreme cases over long spans of pieces — to unite disparate regions of a score. While the *Gestalt* mechanism that tells a reader that notes "belong" to a slur may not lead her to desire uniform spacing across line breaks, other groups, like beams, have a strong grouping effect on the notes they contain because of the way they shape how performers cognitively assimilate musical information. Cole [5] writes:

> When we read our own language, we recognize familiar word-profiles, pick up clues as to function and structure of sentences from characteristic successions of letters or words. We recognize grammatical situations as we recognize faces – not by this or that detail, but on an over-all basis. The same process applies in music reading.... One of the great advantages of the graphical features on the notational map is the advance warning that we get of the approximate density, direction, and type of movement from a single glance at the bars ahead (pp. 23–26).

It is thus natural that we would want different regions of the same beam, which may cut across several line breaks, to be as uniform as possible in their representation of the "densities" about which Cole speaks.

In this context, the relationship between graphical notation and a piece-wide Spring-Rod algorithm takes on vital importance. If a composer creates a graphic that is supposed to hit certain target points — or span — over a piece that spans multiple lines, one would assume that the composer would want consistent spacing rules that controlled the horizontal layout at all of these target points. Otherwise, the composer would have difficulty estimating how much staff space the graphic would occupy over its duration, and furthermore, slight changes to the content of a line would potentially result in appreciable changes in the layout of a graphic that spanned several lines. By using the horizontal spacing algorithm presented above to smoothly stretch and contract graphical notation, one can engrave images that are spanned across large regions of a score without sacrificing the traditional horizontal spacing from which performers glean crucial musical information.

3. THE GRAPHICAL NOTATION ALGORITHM

This section will present how a graphical notation algorithm, by horizontal spacing data, can smoothly engrave graphics in scores. It will first develop a linear-program

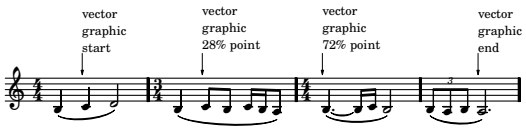

Figure 3. Score fragment to which graphical notation will be appended.

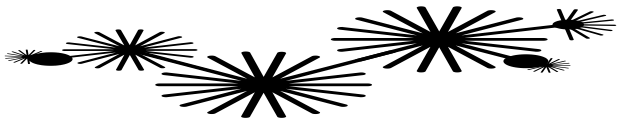

Figure 4. Graphic to be appended.

that effectuates this typesetting, then showing several examples of this algorithm as applied to musical examples.

3.1. The Graphical Notation Linear Program

In order to frame the algorithm proposed in this subsection, consider the following problem. A composer, working with the score fragment in Figure 3, wants to superpose the graphic in Figure 4 such that it hits the target points specified in Figure 5.[3] That is, the vector graphic should have exhausted 28% of its width when it reaches the left end of the C♮ on beat two of the second measure, and it should have exhausted 72% of its width when it reaches the left end of the B♮ on beat one of the third measure. Absent of any spacing algorithm, the composer could simply cut the graphic at these points, stretch the pieces to hit their targets, and obtain a result such as that in Figure 6. One will notice that the different regions of this graphic are scaled using visibly different stretch factors, and even more problematically, awkward kinks are introduced into the figure at the borders established by the target points.

The solution to the dilemma articulated above is formulated as the following linear program. The program takes the standard form

$$\begin{aligned} &\text{maximize } c^{\mathrm{T}} x \\ &\text{subject to } Ax \leq b \end{aligned} \tag{1}$$

[3] All graphics in this paper are assumed to be vector graphics comprised of lines and Bézier curves.

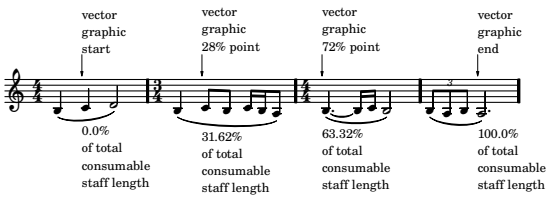

Figure 5. Target points in the original score. Note that the requested distances between the targets do not match up with the actual staff space consumed, thus necessitating a stretching function for the graphic.

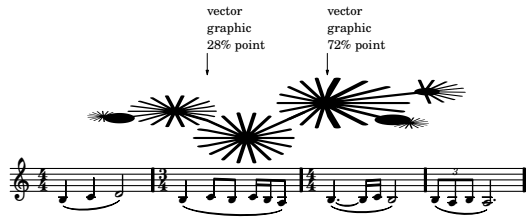

Figure 6. Naive stretching mechanism that changes stretching and contracting factors at target points, giving an uneven look across the graphic and awkward kinks at the target points.

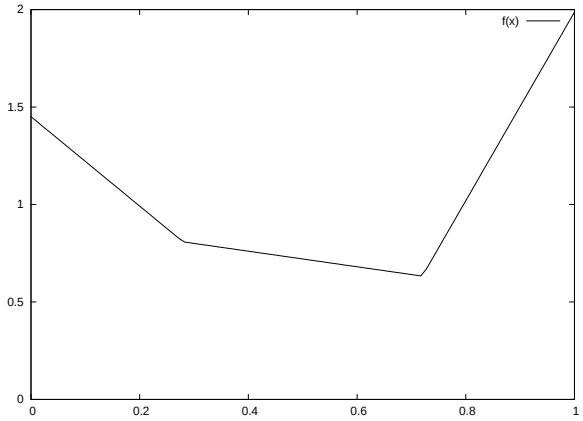

Figure 7. Acceptable smoothing polynomial for the musical example presented in Figure 3.

where the upper rows of matrix A represent equations describing a piecewise linear polynomial that is henceforth called a "smoothing polynomial." The domain of the polynomial represents the length of the vector graphic (normalized from 0 to 1), and the integral of the polynomial represents staff space consumed. This means that, by definition, the Y value of the function at any point is the rate of change of the consumed staff space ($\frac{\Delta \text{ staff space}}{\Delta \text{ graphic length}}$). The goal, then is to have the pieces of the polynomial constrained to touch each other at target points, thus ensuring that there is no disruption in the rate at which the graphic is changing. For example, an acceptable solution to achieve the spacing values requested in Figure 3 would be the one shown in Figure 7. This solution has the function:

$$f(x) = \begin{cases} -2.29378561x + 1.4505339 & x \leq 0.28 \\ -0.39902167x + 0.92 & 0.28 < x \leq 0.72 \\ 4.8365928x - 2.84964242 & 0.72 < x \end{cases} \quad (2)$$

One can further verify that the integral of the function over the [0,1] interval yields the appropriate staff spaces reported in Figure 5.[4]

$$\int_{0.0}^{0.28} (-2.29378561x + 1.4505339)\,dx = 0.3162 \quad (3)$$

$$\int_{0.28}^{0.72} (-0.39902167x + 0.92)\,dx = 0.3170 \quad (4)$$

$$\int_{0.72}^{1.0} (4.8365928x - 2.84964242)\,dx = 0.3668 \quad (5)$$

This solution to the spacing problem is not the only possible one. Multiple potential solutions stem from the fact that there are less constraining equations (five (5) in total – three (3) for the area under the pieces of the polynomial, and two (2) to set the polynomials equal at the target points) than there are variables in the equations that describe the polynomial (six (6) – two (2) for each of the (3) linear equations $y = mx + b$). This mismatch means that certain solutions will force the polynomial below 0, as seen in Figure 8 and verified in Equations 6–9 below:

$$f(x) = \begin{cases} -17.63092847x + 3.5977339 & x \leq 0.28 \\ 9.36097833x - 3.96 & 0.28 < x \leq 0.72 \\ -10.32107333x + 9.24287473 & 0.72 < x \end{cases} \quad (6)$$

$$\int_{0.0}^{0.28} (-17.63092847x + 3.5977339)\,dx = 0.3162 \quad (7)$$

$$\int_{0.28}^{0.72} (9.36097833x - 3.96)\,dx = 0.3170 \quad (8)$$

$$\int_{0.72}^{1.0} (-10.32107333x + 9.24287473)\,dx = 0.3668 \quad (9)$$

Anywhere the polynomial accumulates negative area, the graphic consumes "negative" staff space, or regresses from right to left. Preventing the polynomial from falling below 0 is the reason that one needs to use a linear program (whose constraint takes the form $Ax \leq b$) as opposed to a LaGrangian optimization (whose constraint takes the form $Ax = b$). In the present linear program, the lower rows of matrix A represent the inequalities necessary to assure that each polynomial in the piecewise polynomial stays above 0 at the target points, which guarantees via the intermediate value theorem that the polynomial will be positive at all points in the bounded region.

What follows is a simplified version of $Ax \leq b$ in the form of two equations and one inequality for target points $X = [0, x_1, \ldots, x_{i-1}, 1]$, staff spaces $Y = [0, y_1, \ldots, y_{i-1}, 1]$,[5] and slopes and y-intercepts of a linear piecewise polynomial ${}_M R_B = [(m_0, b_0), \ldots, (m_i, b_i)]$ whose boundaries along the X-axis are the points in set X. Equation 11 represents the constraint that adjacent pieces of the polynomial be

[4] Note that the staff space points from Figure 5 are normalized to fall between 0 and 1. For example, the second integral's result represents the staff space consumed between the 66.32% and 31.62% points ($\frac{63.32-31.62}{100} = 0.3170$).

[5] 0 and 1 bookend the x and y series of values to represent the initial and terminal value for target points and staff spaces respectively.

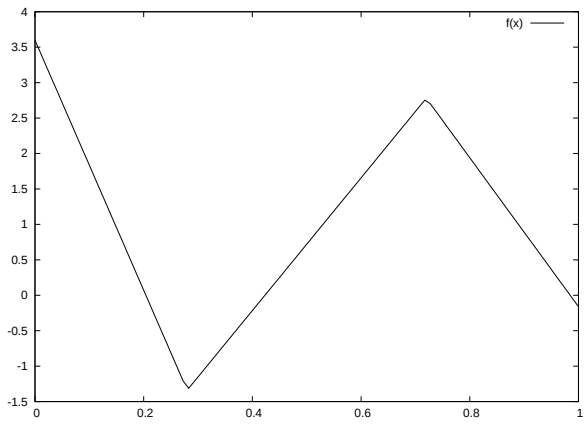

Figure 8. Unacceptable smoothing polynomial for the musical example presented in Figure 3. It attains the same area under the curve as the acceptable solution presented in Figure 7, but accumulates negative area.

connected. Equation 12 assures that the area under each piece of the polynomial is equal to the appropriate amount of staff space to be consumed between two target points. Note that

$$g(a,b) = \frac{a^2 - b^2}{2} \tag{10}$$

Equation 13 guarantees that the piecewise polynomial is positive at each target point. In order for these relations to be turned into a linear program, the equality constraints would have to be rewritten as inequalities using slack variables. Numerous articles and textbooks explain how these variables may be introduced [13][19].

$$\begin{bmatrix} x_1 & 1 & -x_1 & -1 & \dots & 0 & 0 & 0 & 0 \\ \vdots & & & & & & & & \vdots \\ 0 & 0 & 0 & 0 & \dots & x_{i-1} & 1 & -x_{i-1} & -1 \end{bmatrix} \begin{bmatrix} m_0 \\ b_0 \\ \vdots \\ m_i \\ b_i \end{bmatrix}$$

$$= \begin{bmatrix} 0 \\ \vdots \\ 0 \end{bmatrix} \tag{11}$$

$$\begin{bmatrix} g(x_1,0) & x_1 - 0 & \dots & 0 & 0 \\ \vdots & & & & \vdots \\ 0 & 0 & \dots & g(1,x_{i-1}) & 1 - x_{i-1} \end{bmatrix} \begin{bmatrix} m_0 \\ b_0 \\ \vdots \\ m_i \\ b_i \end{bmatrix}$$

$$= \begin{bmatrix} n_1 - 0 \\ \vdots \\ 1 - n_{i-1} \end{bmatrix} \tag{12}$$

$$\begin{bmatrix} 0 & -1 & \dots & 0 & 0 \\ -x_1 & -1 & \dots & 0 & 0 \\ & & & & \\ \vdots & & & & \vdots \\ 0 & 0 & \dots & -1 & -1 \end{bmatrix} \begin{bmatrix} m_0 \\ b_0 \\ \vdots \\ m_i \\ b_i \end{bmatrix}$$

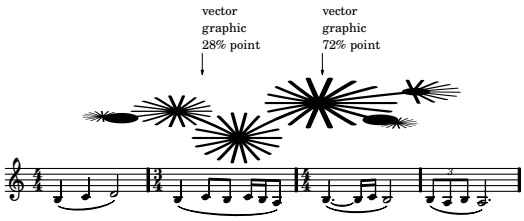

Figure 9. The smoothing linear program applied to the graphic in Figure 6. Note that the graphic hits its target points in a much smoother fashion.

$$\leq \begin{bmatrix} 0 \\ \vdots \\ 0 \end{bmatrix} \tag{13}$$

The optimization portion of the linear program — maximize $c^{\mathrm{T}} x$ — can be designed independent of the constraint matrix described above, and several approaches are possible. For example, one could minimize the average distance that any given point in the graphic is from where it would have originally been if it were not stretched to hit intermediary target points. However, the problem with this approach is that it does not guarantee the avoidance of local anomalies around the target points, which is what this algorithm is supposed to eliminate. Thus, in my implementation of the linear program, I have chosen instead to minimize the *sum of the absolute value of the difference in slope of adjacent line segments*. Using the slopes (m) from Equations 11–13, this can be expressed as

$$\min(\sum_{n=0}^{i-1} |m_{n+1} - m_n|) \tag{14}$$

Or, in other words, minimize the extent to which the slope changes at the target points. By pushing error away from the target points and over the entire graphic, the distortion is less noticeable on a local level without suffering appreciably on the whole, as seen in Figure 9.

The smoothing polynomial does not need to confine itself to interpolation at target points. By further splitting up the piecewise polynomial into smaller lines, it can better approximate curves of higher degrees, thus guaranteeing smoother transitions between rates of change in the stretching of the graphic. In some cases this added detail is the only way to find an acceptable solution. To achieve the results in Figure 10, the first example must use three times as many line segments in its smoothing polynomial to arrive at a successful approximation, whereas the second must use four times as many. Not surprisingly, in these more computationally demanding situations, the success of the algorithm is more apparent when compared to an untreated graphic, as seen in Figure 11.

3.2. Alternative Approaches and Criticism

In order to achieve smooth derivatives along the piecewise polynomial (and thus smoother interpolation at the target points), it would be necessary to use polynomials

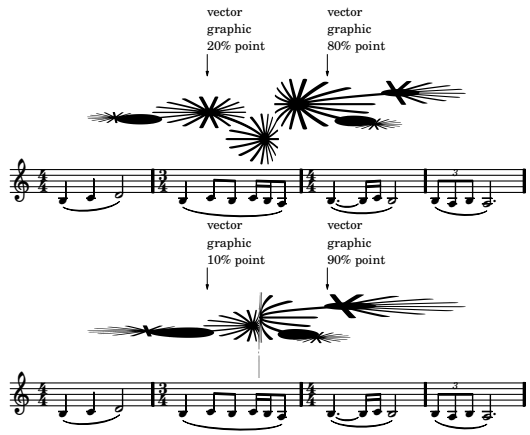

Figure 10. By adding more segments to the piecewise smoother polynomial, more severe stretching scenarios can be handled smoothly.

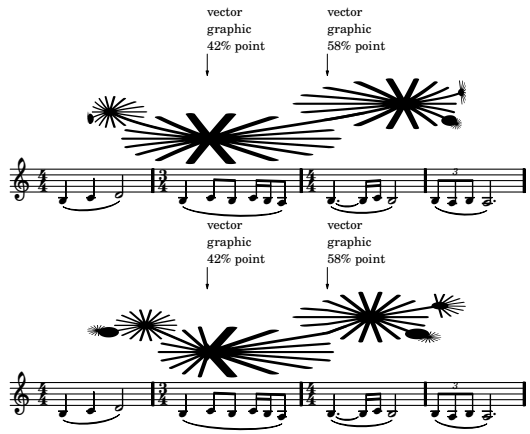

Figure 11. The top graphic is smoothed by a linear approximation of a quartic polynomial to avoid regressing. Below is the same graphic without the algorithm applied. One can see that the unevenness and kinks are more pronounced that those of Figure 6, whose targets require less stretching.

of higher degrees. As stated above, one manner of doing this is by approximating non-linear curves with line segments. Another possibility is the use of semidefinite programming to create positive polynomials of arbitrarily high degrees. If a polynomial is verified to be the sum of squares over a region, then all of its values in that region will be positive. Because sum-of-squares polynomials can be expressed in the form $x^{\mathrm{T}}Ax$, which will yield a real symmetric matrix with nonnegative eigenvalues, their coefficients can be solved for by traversing matrices on the semidefinite cone [20]. While this approach can lead to smoother results, it suffers from a number of computational and practical problems. First, matrices of similar density take longer to solve in semidefinite than linear programming. Second, approximating non-linear polynomials leads to a greater chance of rounding error than working with a group of line segments. Lastly, and perhaps most importantly, any greater accuracy that semidefinite programming affords is not visible at the orders of resolution at which typeset music is usually printed.

Irrespective of the algorithm used, a criticism of this approach is that it can distort the graphic so severely as to denature the original aesthetic statement that the composer intended to make with the image. It is true that, in using this algorithm, severe stretching between target points may cause the composer to reconsider a given graphic, and only below a certain threshold of visual tolerance (i.e. Figure 9) will one fail to immediately notice the stretching being applied to the image. However, because of the algorithm's speed and ability to be invoked directly from a markup language,[6] the composer's decision to modify a source graphic can be made quickly with respect to the evolving visual output. Additionally, insofar as graphics provide an opportunity for variation and transformation that is akin to that which one would find in musical motivic manipulation, this algorithm permits an evolution of graphical objects over scores that would be painstaking to calculate, make, and remake by merging music and graphics in a vector graphics editor. Lastly, for objects whose graphical integrity depends on their relative spacing with respect to noteheads (i.e. waveforms and spectrograms), this linear program provides a manner to preserve the alignment of arrival points without sacrificing the score's horizontal layout.

3.3. Musical Examples

Having established the manner in which horizontal spacing data can be harvested by a linear program to create smoothing polynomials, I will now show some of the added advantages that come from this approach in more developed musical contexts.

As mentioned in the first section, a major concern of the horizontal spacing algorithm in [15] is preserving continuity of staff spacing over line breaks. Using data from

[6]The musical figures in this paper, for example, were made using GNU Lilypond [15]. The linear programming solver is the simplex method [6] as implemented by the GNU Linear Programming Kit [12].

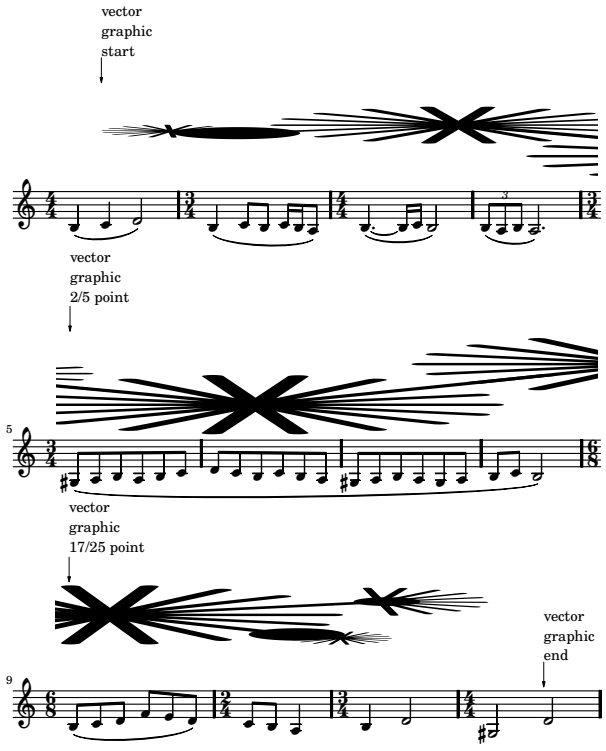

Figure 12. The vector graphic from above is not distorted at the line break because of the continuity in horizontal spacing data.

this algorithm to create a smoothing polynomial allows graphical notation to span over line breaks without altering its rate of spacing on either side of the break, as seen in Figure 12. Furthermore, because horizontal spacing calculations factor in data from all staves in a system, graphics in different staves that have similar horizontal target points will line align vertically as seen in Figure 13. Lastly, the same linear program implemented in the horizontal domain is implemented in the vertical domain in Figure 14, achieving a constancy of error distribution along the Y-axis that further contributes to the visual unity of the score.

4. CONCLUSION

This paper has proposed a model for the spanning of graphical notation that uses data from canonical horizontal spacing algorithms. By feeding this data into a linear program that distributes stretching error over a graphic while preventing right-to-left regressions, one can take a jolty graphic (see Figure 6) and make it acceptably smooth (see Figure 9). Because this approach is programmatically integrated into score markup languages, it can easily control the distribution of graphics over multiple regions of a score. It is perhaps this aspect of the graphical notation linear program that is most promising for future research. As stated in the introduction, a disadvantage of using traditional horizontal spacing mechanisms is that they limit the scope of experimentation one can do with graphics.

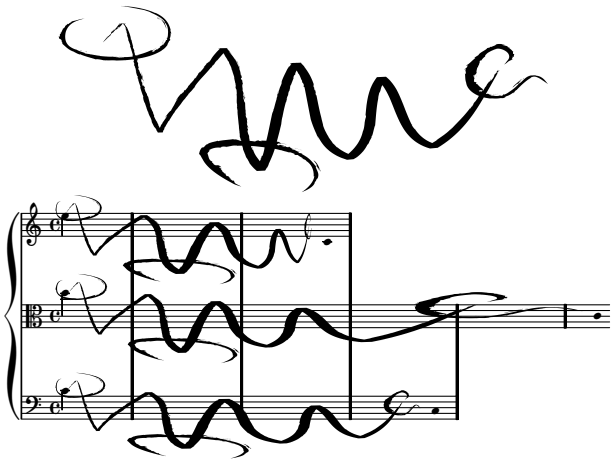

Figure 13. A target point that aligns the lacuna of the glissando figure (presented in its original form at the top of the score) so that it arrives at the same place in all three voices.

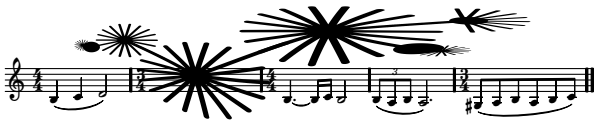

Figure 14. A smoothing polynomial guides the graphic along the Y-axis to intersect with the staff.

However, by generating fragments of traditional score material that are governed by and even fed into the graphical notation algorithm, one can eventually create the bending staves and curving beams that are characteristic of more ambitious experiments such as Burnson's *Bike Ride* or those found in Crumb's *Makrocosmos*. Doing so can achieve a sense of traditional horizontal proportionality in an otherwise advanced graphical aesthetic.

5. REFERENCES

[1] P. Bellini and P. Nesi, "Automatic Justification and Line-Breaking of Music Sheets," *International Journal of Human-Computer Studies*, vol. 61, pp. 104–137, July 2004.

[2] D. Blostein and L. Haken, "Justification of Printed Music," *Communications of the ACM*, vol. 33, pp. 88–99, March 1991.

[3] W. A. Burnson, "Belle, Bonne, Sage: The Beautiful, Good, Wise C++ Vector-Graphics Library for Music Notation," 2010. [Online]. Available: http://bellebonnesage.sourceforge.net

[4] D. Byrd, "Music Notation by Computer," Ph.D. dissertation, Indiana University, 1984.

[5] H. Cole, *Sounds and Signs*. London: Oxford University Press, 1974.

[6] G. Dantzig, *Linear Programming and Extensions*. Princeton: Princeton University Press, 1963.

[7] M. Gieseking, "Code-basierte Generierung interaktiver Notengraphik," Ph.D. dissertation, Universität Osnabrück, 2001.

[8] J. S. Gourlay, "Spacing a Line of Music," Department of Computer and Information Science, The Ohio State University, Tech. Rep. OSU-CISRC-10/87-TR35, 1987.

[9] L. Haken and D. Blostein, "A new algorithm for horizontal spacing of printed music," *Proceedings of the International Computer Music Conference*, pp. 118–119, 1995.

[10] H. H. Hoos, "The GUIDO Notation Format: A Novel Approach for Adequately Representing Score-Level Music," *Proceedings of the International Computer Music Association*, pp. 451–454, 1998.

[11] D. E. Kunth and M. F. Plass, "Breaking Paragraphs into Lines," *Software – Practice and Experience*, vol. 11, pp. 1119–1184, November 1981.

[12] A. Makhorin, "GNU Linear Programming Kit, version 4.45," 2010. [Online]. Available: http://www.gnu.org/software/glpk

[13] H. Nabli, "An Overview on the Simplex Algorithm," *Applied Mathematics & Computation*, vol. 210, no. 2, pp. 479–489, 2009.

[14] H.-W. Nienhuys, "Re: ICMC Paper," Electronic Mail, January 2011.

[15] H.-W. Nienhuys, J. Nieuwenhuizen *et al.*, "Lilypond, the GNU Music Typesetter, version 2.13.47," 2011. [Online]. Available: http://www.lilypond.org

[16] G. Read, *Music Notation, A Manual of Modern Practice*. Boston: Allyn and Bacon, 1964.

[17] K. Renz, "An improved algorithm for spacing a line of music," *Proceedings of the International Computer Music Conference*, pp. 475–481, 2002.

[18] T. Ross, *The Art of Music Engraving and Processing: A Complete Manual, Reference and Text Book on Preparing Music for Reproduction and Print*. Miami: Hansen Books, 1970.

[19] R. I. Rothenberg, *Linear Programming*. New York: North Holland, 1979.

[20] C. W. Scherer and C. W. J. Hol, "Matrix Sum-of-Squares Relaxations for Robust Semi-Definite Programs," *Mathematical Programming*, vol. 107, no. 1/2, pp. 189 – 211, 2006.